I590 – Python Spring 2020

Assignment No:	10	Released:	March 15
Title:	pandas	Due:	April 5

Note:

Use *pandas* to perform vectorized operations on *DataFrames* and *Series*. Please do not use any built in *pandas, NumPy* or *math* functions that calculate average, variance or standard deviation.

Sample variance and standard deviation

Let *X* and *Y* be samples with *n* elements:

$X = x_0 , x_1 , . . . , x_{n-1}$

$Y = y_0 , y_1 , . . . , y_{n-1}$

Let $\bar{x}$ be the mean (average) of *X,* and $\bar{y}$ the mean (average) of *Y*.

Sample variance of *X* is:

$$var(X) = \frac{\sum_{i=0}^{i=n-1}(x_i - \bar{x})^2}{n-1}$$

Sample standard deviation of *X* is the square root of *var*:

$$std(X) = \sqrt{var}$$

Sample covariance and correlation

Sample covariance measures the strength and the direction of the relationship between the elements of two samples. Sample covariance between *X* and *Y* is:

$$cov(X,Y) = \frac{\sum_{i=0}^{i=n-1}(x_i - \bar{x})\,(y_i - \bar{y})}{n-1}$$

Sample correlation:

$$cor(X,Y) = \frac{cov(X,Y)}{std(X) \cdot std(Y)}$$

Stock data

Functions from “pandas_datareader.data” and “pandas_datareader.wb” extract data from various Internet sources into a pandas *DataFrame*, see:

https://pandas-datareader.readthedocs.io/en/latest/

Run the following statement on command prompt to install the package:

```
pip install pandas-datareader
```

The sample output below shows “IBM” stock data from January 1, 2020 to January 15, 2020. The column “Adj Close” was dropped and the resulting data frame with stock data printed.

```
import pandas as pd
from pandas_datareader import data as web

df = web.DataReader("IBM", 'yahoo', '1/1/2020', '1/15/2020')

df.columns.values
Out[520]: array(['Open', 'High', 'Low', 'Close', 'Adj Close',
'Volume'], dtype=object)

# drop “Adj Close”
df = df.drop(['Adj Close'], axis = 1)

df
Out[525]:
                  Open        High         Low       Close    Volume
Date
2020-01-02  135.919998  134.770004  135.000000  135.419998  3148600
2020-01-03  134.860001  133.559998  133.570007  134.339996  2373700
2020-01-06  134.240005  133.199997  133.419998  134.100006  2425500
2020-01-07  134.960007  133.399994  133.690002  134.190002  3090800
2020-01-08  135.860001  133.919998  134.509995  135.309998  4346000
2020-01-09  136.789993  135.309998  135.740005  136.740005  3730600
2020-01-10  137.869995  136.309998  137.000000  136.690002  3255400
2020-01-13  136.639999  135.070007  135.479996  136.600006  3531000
2020-01-14  137.139999  135.550003  136.279999  135.820007  3681000
2020-01-15  138.059998  135.710007  136.000000  136.619995  4045900
```

Note that the rows of the *DataFrame* contain the date and stock data for the days when the stock exchange was open. When we say “find the max period the stock value was up” we refer to the days when the stock exchange was open.

Use statement:

```
df.to_csv("ibm.csv")
```

to save the data frame to “ibm.csv” file and manipulate data with Excel.

Question 1 (20 points)

Use *pandas* to download IBM stock data for year 2019 into a *DataFrame*, calculate and print the variance of close values using formula:

$$var(Close) = \frac{\sum_{i=0}^{i=n-1}(x_i - \bar{x})^2}{n-1}$$

x_i is the stock close value on day i, $\bar{x}$ is the mean (average) of all stock close values in 2019 and n is the number of rows in the *DataFrame* (equal to number of days the stock exchange was open in 2019). Calculate and print the standard deviation:

$$std(Close) = \sqrt{var(Close)}$$

Find and print max and min closing stock values and the respective dates.

Print all numbers rounded to two decimal places.

Sample output:

```
Program that calculates IBM stock data statistics for year 2019.
There are 252 rows of stock data.

IBM 2016 average stock value:  136.99
IBM stock variance:             38.05
IBM stock standard deviation:    6.17

The max stock value 151.36 was on 2019-07-26
The min stock value 112.91 was on 2019-01-03
```

Hints:

Statement `df.shape` returns the dimensions of *DataFrame* “df”.

Statement `df["Close"]` extracts stock close values (column “Close”) into a *Series*.

The following statements find the max value and the index of the max element of the column “Close”:

```
df["Close"].max()
df["Close"].idxmax()
```

Note that the index in stock data is a *Timestamp* object. Use *date* method to get date from a *Timestamp* “dateMax”:

```
print(dateMax.date())
```

Question 2 (20 points)

Write function “rel_std(stock_data)” which receives the stock data downloaded from a web site, and returns a relative standard deviation (which is standard deviation of the stock close values divided by the stock close value average and multiplied by 100).

$$relstd(Close) = \frac{std(Close)}{\overline{Close}} 100$$

Function “main()”:

- Downloads stock data for year 2019 for 8 Dow Jones companies:

  Boeing, Coca-Cola, Exxon Mobil, General Electric, JPMorgan Chase, Nike, Pfizer, Verizon

- Calls “relStd(sData)” to collect relative standard deviation for the companies.
- Stores relative standard deviations in a dictionary, with company stock symbol as a key and relative standard deviation as value.
- Print the company names and corresponding relative standard deviations. It is not important if the order of the companies is not the same as when you inserted them into the dictionary.

Print all numbers rounded to two decimal places.

Sample output:

```
Program that calculates relative standard deviation
for eight Dow Jones companies.

Company                  Relative std deviation
Boeing                   6.69
Coca Cola                6.43
Exxon Mobil              6.04
General Electric         9.5
JP Morgan                9.14
Nike                     6.49
Pfeizer                  7.14
Verizon                  3.35
```

Hints:

Dictionaries are mutable, slide 5. lecture 10.3 shows how to insert a new “key – value” pair.

You may want to use two dictionaries, one with stock symbols (as keys) and relative standard deviation (as values). The other dictionary contains stock symbols (as keys) and the values that are the company names.

Use “for” loop to Iterate through a dictionary:

```
for key in dic:
    print(dic[key])
```

Question 3 (20 points)

Write function “correl(stock1, stock2)” which receives the stock data of two companies and returns sample correlation between their close values:

$$cor(close1, close2) = \frac{cov(close1, close2)}{std(close1) \cdot std(close2)}$$

Here:

- *cov(close1, close2)* is the sample covariance between stock close values of *stock1* and *stock2*. See page 1 for the sample covariance formula.
- *std(close1)* and *std(close2)* are sample standard deviations of the *stock1* and *stock2*, respectively. See page 1 for the sample standard deviation formula.

Function “main()”:

- Downloads *Boeing* and *Coca-Cola* stock data for 2019
- Calls “correl(stock1, stock2)” to get and print the sample correlation between the close values of the two companies.

Print the numbers rounded to two decimal places.

Sample output:

```
Program calculates sample correlation between Boeing and Coca Cola
closing stock values in 2019.


Average Boeing stock value:    365.03
Std deviation of Boing stock:   24.4

Average Coke stock value:       50.83
Std deviation of Coke stock:     3.27

Correlation between BA and KO:  -0.49
```

Question 4 (20 points)

Write function “max_up(stock)” that receives a *DataFrame* “stock” with stock data and finds out the maximum period in which the close value of the stock was up each day. Consider the stock up if the close value is greater than the close value of the previous day.

Function returns the number of (working) days in the period and index of the period start.

Function “main()”:

- Prompts the user for the stock symbol and downloads the stock data for the year 2019 into a *DataFrame* “df”.
- Calls function “max_up(df)” to get the number of days of the max period the stock was up each day, and the index of the period start.
- Prints the period length in days, date of period start and period end. (“days” refer to the days the stock exchange was open - the days listed in the *DataFrame* “df”.)

Sample output:

```
Program that finds the longest period a stock was up in 2019.

Please enter the stock symbol: ibm

The longest up period for the stock symbol IBM:
Length in days:             7
Period started on:          2019-03-11
Close stock value at start: 137.71
Period ended on:            2019-03-19
Close stock value at end:   140.49
```

Hints:

You are able to index the rows with integers. For example, the following “for” loop prints the close stock value if it is up compared to the close value of the previous day:

```
for i in range(1, len(df)):
    if df["Close"][i] > df["Close"][i-1]:
        print(df["Close"][i])
```

Use counter pattern to count the days the stock was up.

Function “max_up(df)” returns “max” (the number of working days of the max period the stock was up each day) and “imax” (the index of the period start).

Get the start *Timedate* of the period start with the statement:

```
start_date = df["Close"].index[imax]
```

The close value of the stock on the “start_date”:

```
df["Close"][start_date]
```

Question 5 (20 points)

The regression line is the line with the following equation:

$$y = \bar{y} + k\,(x - \bar{x})$$

where:

$$k = \frac{\sum x_i y_i - m\,\bar{x}\bar{y}}{\sum x_i^2 - m\,\bar{x}^2}$$

$\bar{x}$ is the mean of the *x*-values, $\bar{y}$ is the mean of the *y*-values, and *m* is the number of points.

Write a program that uses *pandas* to plot a regression line, that is, the line with the best fit through a collection of points. First, ask the user to specify the data points by clicking on them in a graphics window. To find the end of input, place a small rectangle labeled “Draw” in the lower left corner of the window; the program will stop gathering points when the user clicks inside the rectangle.

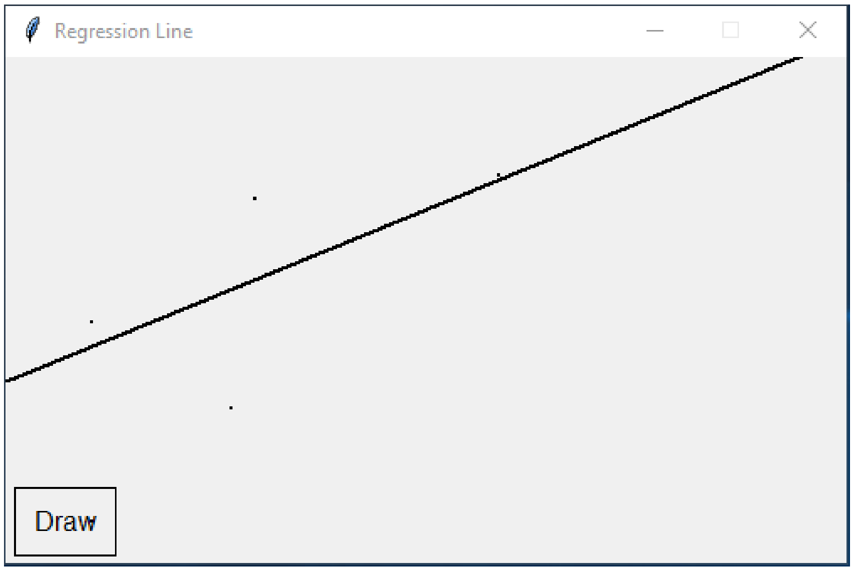
Draw all points on the graphics window.

Collect the points in a dictionary; convert the dictionary to a *DataFrame*, and use *pandas* to perform computation without using loops.

Use *pandas* “sum” function; do not use *pandas* function for average.

If the user enters no point or one point, program prints a message “You need at least two points for a regression line” and terminates.

Sample output:



Values of program variables for the previous example:

```
Dictionary with points:
{0: [50.0, 156.0], 1: [133.0, 207.0], 2: [147.0, 83.0], 3: [292.0,
69.0]}

DataFrame with points:
       X      Y
0   50.0  156.0
1  133.0  207.0
2  147.0   83.0
3  292.0   69.0

Number of points:  4
Average of x:      155.5
Average of y:      128.75
Numerator of k:    -12402.5
Denominator of k:  30341.0
Slope k:           -0.4087703108005669
Point 1:
   x1 =            0
   y1 =            192.31378332948816
Point 2:
   x2 =            500
   y2 =            -12.071372070795292
```

Hints:

As the user clicks on points, the program should draw them in the graphics window and collect the points into a dictionary. When the user clicks inside "Draw" rectangle, the program converts the dictionary to a *DataFrame*, and uses *pandas* operations to calculate $\bar{x}$, $\bar{y}$ and *k*.

Add point "p" to a dictionary "points":

```
points[i] = [p.getX(), p.getY()]
```

Convert the dictionary to a *DataFrame* "df":

```
df = DataFrame(dPoints)
```

You will get a *DataFrame* "df" with point in columns. I suggest to transpose "df" to get points in rows:

```
points = df.T
```

DataFrame "points" contains the coordinates *x* and *y of* all entered points in columns *X* and *Y*, respectively. Use vectorized operations to calculate mean of *x* and *y* and calculate slope *k*.

You will need two points to draw a line. The first point is for $x_1 = 0$ (which is the left border of the graphics window) and the second point is for $x_2 = 500$ (which is the right border of the graphics window.

Find the corresponding y_1 and y_2 coordinates of the two points by using the equation:

$$y = \bar{y} + k\,(x - \bar{x})$$

Draw the regression line in the graphics window with the statements:

```
reg_line = Line(Point(x1,y1), Point(x2,y2))
reg_line.draw(win)
```

Once you draw the regression line, wait for a mouse click, close the graphics window and terminate the program.

Instructions

- Write your programs in Python 3. Write a separate program for each question.
- Make sure your code is properly formatted, structured and commented. Include a header to program file with your name, date, question number and short program description.
- Do your best to follow the programming style suggested in *Python Enhancement Proposal PEP8* (outlined in a tutorial https://realpython.com/python-pep8/). Follow the naming conventions for variables and functions.
- Make sure that your program compiles. A program that doesn't compile is usually scored 0 points.
- Each program should be stored in a separate Python file. Please name your files such that it starts with the word "question" and ends with the number of the question. All files should be executable python files (they should all end in .py). Acceptable examples include *Question_5.py*, *question_4.py*, *Question3.py*, and *question2.py.*
- A program should run the entire code for a particular question by directly executing it. Include call to `main()` function at the end of your program. Provide any data files that your program needs to execute.
- Please upload to Canvas separate files for each question. If there are five questions, you will be uploading five separate files.
- If you want to provide more instructions for executing your code, upload a 'readme.txt' file, where you may provide additional information.